Sound, Part 2

Using range to manipulate samples by index number

Knowing where we are in the sound

- More complex operations require us to know where we are in the sound, which sample
  - Not just process all the samples exactly the same
- Examples:
  - Reversing a sound
    - It's just copying, like we did with pixels
  - Changing the frequency of a sound
    - Using sampling, like we did with pixels
  - Splicing sounds

Increasing volume by *sample index*

```
def increaseVolumeByRange(sound):
   for sampleNumber in range(1, getLength(sound) + 1):
     value = getSampleValueAt(sound, sampleNumber)
     setSampleValueAt(sound, sampleNumber, value * 2)
```

This really is the same as:

```
def increaseVolume(sound):
   for sample in getSamples(sound):
     value = getSample(sample)
     setSample(sample,value * 2)
```

Recipe to play a sound backwards (Trace it!)

```
def playBackward(filename):
   source = makeSound(filename)
   dest = makeSound(filename)

   srcIndex = getLength(source)
   for destIndex in range(1, getLength(dest) + 1):
     srcSample = getSampleValueAt(source, srcIndex)
     setSampleValueAt(dest, destIndex, srcSample)
     srcIndex = srcIndex - 1

   return dest
```

Start at end of sound

Work backward

Return the processed sound for further use in the function that calls playBackward

Walkthrough

```
def playBackward(filename):
   source = makeSound(filename)
   dest = makeSound(filename)

   srcIndex = getLength(source)
   for destIndex in range(1, getLength(dest) + 1):
      srcSample = getSampleValueAt(source, srcIndex)
      setSampleValueAt(dest, destIndex, srcSample)
      srcIndex = srcIndex - 1

   return dest
```

12	25	13

source

12	25	13

dest

How does this work?

- We make two copies of the sound
- The **srcIndex** starts at the end, and the **destIndex** goes from 1 to the end.
- Each time through the loop, we copy the sample value from the **srcIndex** to the **destIndex**

Note that the **destIndex** is *increasing* by 1 each time through the loop, but **srcIndex** is *decreasing* by 1 each time through the loop

```
def playBackward(filename):
   source = makeSound(filename)
   dest = makeSound(filename)

   srcIndex = getLength(source)
   for destIndex in range(1, getLength(dest) + 1):
      srcSample = getSampleValueAt(source, srcIndex)
      setSampleValueAt(dest, destIndex, srcSample)
      srcIndex = srcIndex - 1

   return dest
```

Uses?

- Just for fun
- Sound reversals in music, speech, etc…

Alternate Version

- Remember this pseudocode to flip an image?

```
for y in the range 1 to imageHeight
  for x in the range 1 to imageWidth / 2
     pixelLeft = pixel at coordinate (x,y)
     pixelRight = pixel at coordinate (imageWidth - x + 1, y)
     swap the colors of pixelLeft and PixelRight by:
          colorLeft = getColor(pixelLeft)
          colorRight = getColor(pixelRight)
          set the color of pixelLeft to colorRight
          set the color of pixelRight to colorLeft
```

Alternate Version

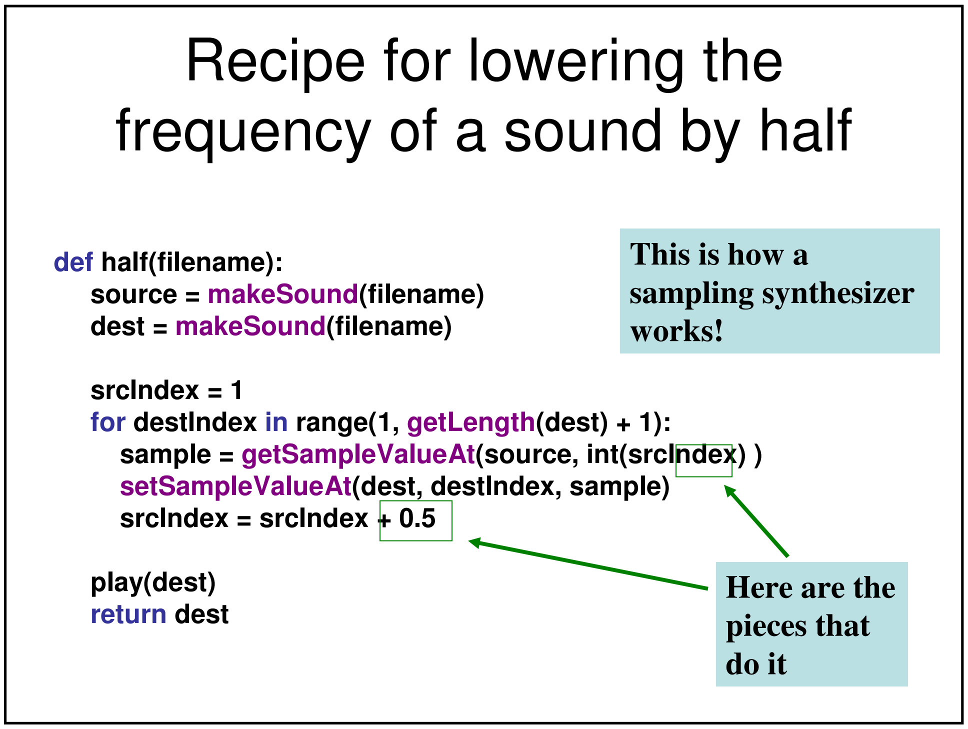
```
def reverseSound(filename):
  source = makeSound(filename)
  for x in range(1, getLength(source) / 2):
    leftPosition = x
    rightPosition = getLength(source) - x + 1
    leftSample = getSampleValueAt(source, leftPosition)
    rightSample = getSampleValueAt(source, rightPosition)
    setSampleValueAt(source, leftPosition, rightSample)
    setSampleValueAt(source, rightPosition, leftSample)

  play(source)
  return source
```

12	25	13	41	11	49

source

Changing Sound Frequencies

- Higher frequency interpreted as higher pitch
  - If the sampling rate stays the same this can be accomplished by eliminating samples
  - E.g. eliminate every other sample to “half” the sound
- Lower frequency interpreted as lower pitch
  - If the sampling rate stays the same this can be accomplished by duplicating samples
  - E.g. “Double” the sound by having each sample appear twice

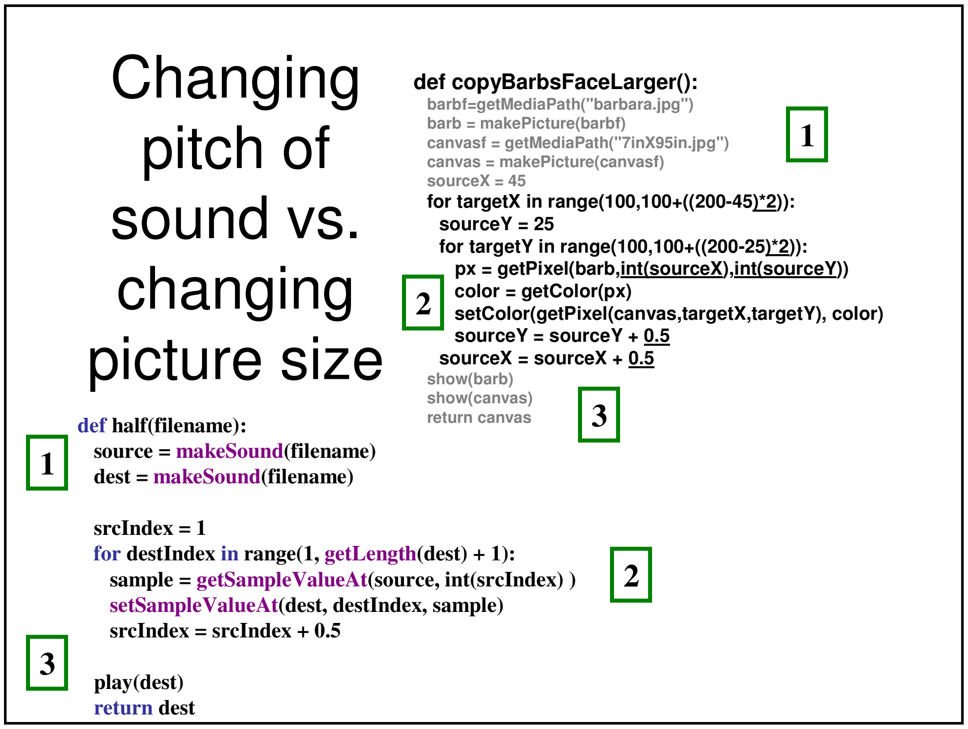


Recipe for lowering the frequency of a sound by half

```
def half(filename):
    source = makeSound(filename)
    dest = makeSound(filename)

    srcIndex = 1
    for destIndex in range(1, getLength(dest) + 1):
       sample = getSampleValueAt(source, int(srcIndex) )
       setSampleValueAt(dest, destIndex, sample)
       srcIndex = srcIndex + 0.5

    play(dest)
    return dest
```

This is how a sampling synthesizer works!

Here are the pieces that do it



Changing pitch of sound vs. changing picture size

```
def copyBarbsFaceLarger():
  barbf=getMediaPath("barbara.jpg")
  barb = makePicture(barbf)
  canvasf = getMediaPath("7inX95in.jpg")
  canvas = makePicture(canvasf)
  sourceX = 45
  for targetX in range(100,100+((200-45)*2)):
    sourceY = 25
    for targetY in range(100,100+((200-25)*2)):
      px = getPixel(barb,int(sourceX),int(sourceY))
      color = getColor(px)
      setColor(getPixel(canvas,targetX,targetY), color)
      sourceY = sourceY + 0.5
    sourceX = sourceX + 0.5
  show(barb)
  show(canvas)
  return canvas
```

1 2 3

```
def half(filename):
  source = makeSound(filename)
  dest = makeSound(filename)

  srcIndex = 1
  for destIndex in range(1, getLength(dest) + 1):
    sample = getSampleValueAt(source, int(srcIndex) )
    setSampleValueAt(dest, destIndex, sample)
    srcIndex = srcIndex + 0.5

  play(dest)
  return dest
```

1 2 3

Both of them are *sampling*

- Both of them have three parts:
  1. Initialization - objects are set up
  2. A loop where samples or pixels are copied from one place to another
     - To decrease sound frequency or increase image size, we take each sample/pixel twice
     - In both cases, we do that by incrementing the source index by 0.5 instead of 1 and taking the integer of the index
  3. Finish up and return the result

Recipe to double the frequency of a sound

Here's the critical piece: We skip every other sample in the source!

```
def double(filename):
  source = makeSound(filename)
  target = makeSound(filename)
  targetIndex = 1
  for sourceIndex in range(1, getLength(source) + 1, 2):
      value = getSampleValueAt(source, sourceIndex)
      setSampleValueAt( target, targetIndex, value)
      targetIndex = targetIndex + 1
  #Zero out the rest of the target sound -- it's only half full!
  # Zeros are silent.
  for secondHalf in range( getLength( target)/2, getLength( target) - 1):
      setSampleValueAt(target, targetIndex, 0)
      targetIndex = targetIndex + 1
  play(target)
  return target
```

What happens if we don’t "zero out" the end?

Try this out!

```
def double(filename):
  source = makeSound(filename)
  target = makeSound(filename)
  targetIndex = 1
  for sourceIndex in range(1, getLength(source)+1, 2):
     value = getSampleValueAt(source, sourceIndex)
     setSampleValueAt( target, targetIndex, value)
     targetIndex = targetIndex + 1
  #Clear out the rest of the target sound -- it's only half full!
  #for secondHalf in range( getLength( target)/2, getLength( target) - 1):
  #  setSampleValueAt(target,targetIndex,0)
  #  targetIndex = targetIndex + 1
  play(target)
  return target
```

“Switch off” these lines of code by commenting them out.

Splicing Sounds

- Splicing gets its name from literally cutting and pasting pieces of magnetic tape together
- Easy to do in a program if each sound is in its own file

Merging Separate Sounds

```
def merge():
  kenricksound = makeSound("kenrick.wav")
  issound = makeSound("is.wav")
  target = makeSound(getMediaPath("sec3silence.wav"))

  index = 1
  # Copy in "Kenrick"
  for src in range(1, getLength(kenricksound)):
    value = getSampleValueAt(kenricksound, src)
    setSampleValueAt(target, index, value)
    index = index + 1

  # Copy in 0.1 second pause (silence)
  for src in range(1, int(0.1 * getSamplingRate(target))):
    setSampleValueAt(target, index, 0)
    index = index + 1

  # Copy in "is"
  for src in range(1, getLength(issound)):
    value = getSampleValueAt(issound, src)
    setSampleValueAt(target, index, value)
    index = index + 1

  play(target)
  return(target)
```

Merging Sounds

- What if we didn’t add the pause?
- What if the sounds were recorded at different volumes, how might we make them match?

Changing the splice

- What if we wanted to increase or decrease the volume of an inserted word?
  - Simple! Multiply each sample by something as it's pulled from the source.
- Could we do something like slowly increase volume (emphasis) or normalize the sound?
  - Sure! Just like we've done in past programs, but instead of working across *all* samples, we work across only the samples in that sound!

Making more complex sounds

- We know that natural sounds are often the combination of multiple sounds.
- Adding waves in physics or math is hard.
- In computer science, it's easy! Simply add the samples at the same index in the two waves:

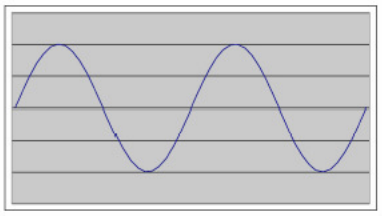
```
for srcSample in range(1, getLength(source)+1):
    destValue = getSampleValueAt(dest, srcSample)
    srcValue = getSampleValueAt(source, srcSample)
    setSampleValueAt(source, srcSample, srcValue+destValue)
```

Adding sounds

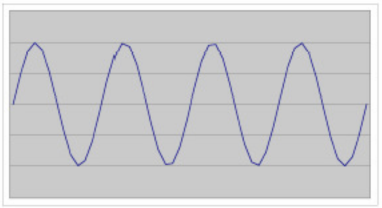
The first two are sine waves generated in Excel.

The third is just the sum of the first two columns.

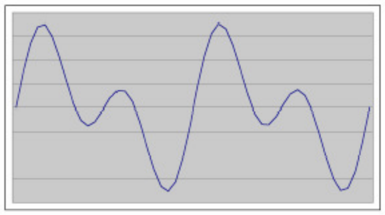
a



b



a + b = c



Uses for adding sounds

- We can mix sounds
  - We even know how to change the volumes of the two sounds, even over time (e.g., fading in or fading out)
- We can create echoes
- We can add sine (or other) waves together to create kinds of instruments/sounds that do not physically exist, but which sound interesting and complex

A function for adding two sounds

```
def addSoundInto(sound1, sound2):

  for sampleNmr in range(1, getLength(sound1)+1):
    sample1 = getSampleValueAt(sound1, sampleNmr)
    sample2 = getSampleValueAt(sound2, sampleNmr)
    setSampleValueAt(sound2, sampleNmr, sample1 + sample2)
```

Notice that this adds sound1 and sound by adding sound1 *into* sound2

Making a chord by mixing three notes

```
>>> setMediaFolder()
New media folder: C:\mediasources\
>>> getMediaPath("bassoon-c4.wav")
'C:\\mediasources\\bassoon-c4.wav'
>>> c4=makeSound(getMediaPath("bassoon-c4.wav"))
>>> e4=makeSound(getMediaPath("bassoon-e4.wav"))
>>> g4=makeSound(getMediaPath("bassoon-g4.wav"))
>>> addSoundInto(e4,c4)
>>> play(c4)
>>> addSoundInto(g4,c4)
>>> play(c4)
```

Adding sounds with a delay

```
def makeChord(sound1, sound2, sound3):
  for index in range(1, getLength(sound1)):
    s1Sample = getSampleValueAt(sound1, index)
    if index > 1000:
      s2Sample = getSampleValueAt(sound2, index - 1000)
      setSampleValueAt(sound1, index, s1Sample + s2Sample)
    if index > 2000:
      s3Sample = getSampleValueAt(sound3, index - 2000)
      setSampleValueAt(sound1, index, s1Sample + s2Sample + s3Sample)
```

-Add in sound2 after 1000 samples

-Add in sound3 after 2000 samples

Note that in this version we're adding into sound1!